Symbolically Synthesizing Small Circuits

Rüdiger Ehlers\textsuperscript{1}  Robert Könighofer\textsuperscript{2}  Georg Hofferek\textsuperscript{2}

\textsuperscript{1}University of California at Berkeley  United States  \textsuperscript{2}Graz University of Technology  Austria

FMCAD 2012 – 24\textsuperscript{th} October 2012
Verification:

\[ G(u = 0 \rightarrow X(v = 1)) \]

- Satisfied
- Not satisfied (+ counter-example)
Verification and synthesis of reactive systems

Synthesis:

\[ G( u = 0 \rightarrow X( v = 1 )) \]

\[ \text{Input} = \{ u, \ldots \} \]
\[ \text{Output} = \{ v, \ldots \} \]

\[ \Rightarrow \]

Realisable

Not realisable
Synthesis in a nutshell

\[ G(u = 0) \rightarrow X(v = 1) \]

\[ + \]

Input = \{u, \ldots\}

Output = \{v, \ldots\}
Synthesis in a nutshell
int main() {
    bool s = 1;
    while (true) {
        bool b = in-u();
        out-v(b | s);
        s = s ⊕ u;
        wait();
    }
}
On general strategies
On general strategies
Encoding general strategies

\[
\overline{u} \rightarrow \quad \text{pre}_0 \land \text{pre}_1 \land u \land v \land \text{pre}_0 \land \text{pre}_1 \rightarrow \text{pre}_0 \land \text{post}_0 \land \text{post}_0 \land \text{pre}_1 \land \text{post}_1 \land \text{post}_1 \land \text{post}_1
\]
Encoding general strategies

\begin{align*}
\overline{u} & \iff \overline{\text{pre}_0} \land \overline{\text{pre}_1} \land \text{pre}_0 \land \text{pre}_1 \\
\text{post}_0 & \iff \text{post}_0 \\
\text{post}_1 & \iff \text{post}_1 \\

\end{align*}
Encoding general strategies

\[ \overline{u} \quad \overline{pre}_0 \land \overline{pre}_1 \land \overline{u} \land v \land pre_0 \land \overline{pre}_1 \]
Encoding general strategies

\[
pre_0 \land pre_1 \land \bar{u} \land v \land pre_0 \land \overline{pre_1}
\]
Building circuits from general strategies

General strategies ...  
- ... represent allowed behaviour of the system to synthesize
- ... are typically given symbolically
- ... are often huge. To have 64 state bits in the BDD is common.

What is a suitable implementation?  
Any implementation that is a specialisation of the general strategy is fine.
Building circuits from general strategies

combinational circuit

clk
I O
S' SS
U W

combinational circuit

I U W O
S S'
clk

R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012 7 / 22
Previous approaches to building the combinational circuit

Experiences in the synthesis domain

- Kukula and Shiple (2000) - huge, deep circuits
- Bañeres et al. (2004) - extremely slow on synthesis problems
- Bloem et al. (2007b) - best approach so far, circuits are still large
- Jiang et al. (2009) - generates larger circuits than the approach by Bloem et al. (2007b)
- ...
Example: AMBA AHB bus arbiter (Bloem et al., 2007b)

![Graph showing the relationship between the number of clients and the number of gates.](image)

- Number of clients: 2 to 15
- Number of gates: 10k to 130k

Symbolically Synthesizing Small Circuits

R. Ehlers, R. Könighofer, G. Hofferek

FMCAD 2012 9 / 22
Example: AMBA AHB bus arbiter (Bloem et al., 2007b)

Symbolically Synthesizing Small Circuits

R. Ehlers, R. Könighofer, G. Hofferek
FMCAD 2012
Our new approach

Main idea

Use computational learning of Boolean functions as main tool for computing small circuit implementations.

Benefits

- We can efficiently benefit from a large number of don’t care cases
- We obtain shallow circuits
Bit-by-bit circuit extraction approach

R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012
Bit-by-bit circuit extraction approach

R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012
Bit-by-bit circuit extraction approach

Onset

Care set

pre₀

pre₁

pre₀

pre₁

u

v

0

1

R. Ehlers, R. Könighofer, G. Hofferek

Symbolically Synthesizing Small Circuits

FMCAD 2012 11 / 22
Bit-by-bit circuit extraction approach

![Diagram showing bit-by-bit circuit extraction approach]
Bit-by-bit circuit extraction approach

Reachable states/positions

Onset

Care set

pre_0
pre_1
post_0
post_1
u
0
1
v
R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012
Boolean formula learning (DNF)

Candidate solution: \( \psi = \neg \text{pre}_0 \wedge \neg u_0 \psi = (\neg \text{pre}_0 \wedge \neg u_0) \vee (\text{pre}_1 \wedge u_1) \)

Onset

Care Set
Candidate solution: $\psi = false$

$\psi = \neg pre_0 \land \neg u_0$

$\psi = (\neg pre_0 \land \neg u_0) \lor (pre_1 \land u_1)$
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg \text{u}_0 \land (\text{pre}_1 \land \text{u}_1) \]
Boolean formula learning (DNF)

Care Set

Candidate solution: $\psi = \neg \text{pre}_0 \land \neg u_0 \psi = (\neg \text{pre}_0 \land \neg u_0) \lor (\text{pre}_1 \land u_1)$
Boolean formula learning (DNF)

Onset

Candidate solution: \( \psi = \neg \text{pre}_0 \land \neg u_0 \psi = (\neg \text{pre}_0 \land \neg u_0) \lor (\text{pre}_1 \land u_1) \)
Boolean formula learning (DNF)

Candidate solution:

$$\psi = \neg \text{pre}_0 \land \neg u_0 \lor (\text{pre}_1 \land u_1)$$
**Boolean formula learning (DNF)**

Candidate solution:

\[ \psi = \text{false} \]
Candidate solution:

$$\psi = \text{false}$$
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = \text{false} \]
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = \text{false} \]
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = \text{false} \]
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = \text{false} \]

<table>
<thead>
<tr>
<th></th>
<th>( u_0 )</th>
<th>( u_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{pre}_0 )</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>( \text{pre}_1 )</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( u_0 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( u_1 )</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>( \text{out} )</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012 12 / 22
Candidate solution:

\[ \psi = \text{false} \]
### Candidate solution:

\[ \psi = \text{false} \]

**Boolean formula learning (DNF)**

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>0</th>
<th>0</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>u₀</strong></td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**pre₀**

**pre₁**

**u₁**
Candidate solution:
\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Candidate solution:

\[
\psi = \neg \text{pre}_0 \land \neg u_0
\]
Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg \text{u}_0 \]
Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Boolean formula learning (DNF)

Candidate solution:

$$\psi = \neg \text{pre}_0 \land \neg u_0$$
Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Candidate solution:
\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Candidate solution:

\[ \psi = \neg \text{pre}_0 \land \neg u_0 \]
Boolean formula learning (DNF)

Candidate solution:

\[ \psi = (\neg \text{pre}_0 \land \neg u_0) \lor (\text{pre}_1 \land u_1) \]
### Candidate solution:

\[ \psi = (\neg \text{pre}_0 \land \neg u_0) \lor (\text{pre}_1 \land u_1) \]
Candidate solution:

$$\psi = (\neg \text{pre}_0 \land \neg u_0) \lor (\text{pre}_1 \land u_1)$$
CDNF learning: Bshouty’s algorithm

**Z-monotone functions**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(b)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg b)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\neg a)</td>
<td>(a)</td>
<td>(\neg a)</td>
<td>(a)</td>
</tr>
</tbody>
</table>

**CDNF Learning Algorithm**

1. Start with the empty function set.
2. For each input variable, add a literal to the function set.
3. For each possible assignment of values to the variables, evaluate the function set.
4. If the function evaluates to true for any assignment, add that assignment to the training set.
5. If the function evaluates to false for any assignment, delete that assignment from the training set.
6. Repeat steps 3-5 until the function set stabilizes.

**Example**

- For \(a = \text{true}\) and \(b = \text{false}\), the function evaluates to true.
- For \(a = \text{false}\) and \(b = \text{true}\), the function evaluates to false.

**Conclusion**

The Z-monotone function for \(a\) and \(b\) is determined by the training set and the following steps.
Overlay example: decomposing a XOR into CDNF

Idea

\[(a \lor b) \land (\neg a \lor \neg b) = \text{Target function}\]
Algorithm

While target function not found:

- Search for **false-positive** $z$ (as in CNF)
  - If found, add a new $z$-based DNF
- Search for **false-negative** $z$ (as in DNF)
  - If found, update all DNFs to accept $z$
Example: learning $a \text{ XOR } b$

Candidate solution

(true)

Operation

Check for false-positive
Example: learning $a \text{ XOR } b$

Candidate solution

$(\text{true})$

Operation

Check for false-positive $\rightarrow \times: ab$
Example: learning \( a \) XOR \( b \)

Candidate solution

\[
\begin{array}{cc}
 b & \neg b \\
 \neg a & a \\
\end{array}
\]

\( (\text{false}) \)

Operation

Check for false-negative
Example: learning $a$ XOR $b$

Candidate solution

\[
\begin{array}{cc}
 b & \neg b \\
\neg a & a \\
\end{array}
\]

\text{(false)}

Operation

Check for false-negative $\rightarrow \times$: $\neg ab$
Example: learning $a \text{ XOR } b$

Candidate solution

Operation
Check for false-positive
Example: learning a XOR b

Candidate solution

\[ \neg a \quad a \]

\[ \neg b \quad b \]

\[ (\neg a) \]

Operation

Check for false-positive \[ \rightarrow \mathbf{X}: \overline{ab} \]
Example: learning $a$ XOR $b$

**Candidate solution**

$$\neg a \quad a \quad (\neg a) \quad \land \quad \neg b \quad b \quad (false)$$

**Operation**

Check for false-negative
Example: learning $a$ XOR $b$

**Candidate solution**

\[
\begin{align*}
&\neg a & a \\
\neg b & & \neg b \\
& (\neg a) & \land & (\text{false}) \\
\end{align*}
\]

**Operation**

Check for false-negative $\rightarrow \times: \neg b a$
Example: learning a XOR b

Candidate solution

\[(\neg a \vee \neg b) \land (a)\]

Operation

Check for false-positive
Example: learning a XOR b

Candidate solution

\[ (\neg a \lor \neg b) \land (a) \]

Operation

Check for false-positive \(\rightarrow \checkmark\)
Example: learning a XOR b

Candidate solution

\[
\begin{align*}
&b \\
&\neg b \\
&\neg a \quad a \\
&\quad (\neg a \lor \neg b) \\
&\quad \land \\
&\quad (a)
\end{align*}
\]

Operation

Check for false-negative
Example: learning a XOR b

Candidate solution

\[
\begin{align*}
\neg a & \quad a \\

\neg b & \quad \neg b
\end{align*}
\]

\[
\begin{align*}
(\neg a \lor \neg b) & \quad \land \\

(a) & \quad a \\

b & \quad b
\end{align*}
\]

Operation

Check for false-negative $\rightarrow \times: \overline{ab}$
Example: learning a XOR b

Candidate solution

$\neg a \lor \neg b \land (a \lor b)$

Operation

Check for false-positive
Example: learning a XOR b

Candidate solution

\[
\begin{align*}
\neg a & \quad a \\
\neg b & \quad \neg b \\
\neg a \lor \neg b & \quad (a \lor \neg b) \\
\land & \quad \land
\end{align*}
\]

Operation

Check for false-positive $\rightarrow \checkmark$
Example: learning $a$ XOR $b$

**Candidate solution**

\[
\begin{align*}
\neg a & \quad a \\
\neg b & \quad b
\end{align*}
\]

\[
(\neg a \lor \neg b) \land (a \lor b)
\]

**Operation**

Check for false-negative
Example: learning \( a \) XOR \( b \)

**Candidate solution**

\[
\begin{array}{cccc}
\neg b & b & \neg b & b \\
\neg a & a & \neg a & a \\
(\neg a \lor \neg b) & \land & (a \lor b)
\end{array}
\]

**Operation**

Check for false-negative \( \rightarrow \) ✔
Example: learning $a$ XOR $b$

**Candidate solution**

\[
\begin{array}{cc}
\neg a & a \\
\neg b & b \\
\end{array}
\]

\[
\begin{array}{cc}
\neg a & a \\
\neg b & b \\
\end{array}
\]

\[
(\neg a \lor \neg b) \land (a \lor b)
\]

**Operation**

No counter-examples? Done!
Experimental Setup

Benchmarks

- **RATSY** – GR(1) synthesis
  - IBM generalised buffer (Bloem et al., 2007b)
  - AMBA AHB Arbiter (Bloem et al., 2007a,b)

- **Unbeast** – bounded synthesis
  - Load balancer (Ehlers, 2010)
  - Various benchmarks by Jobstmann and Bloem (2006)

Implementation

- Uses **CuDD** as BDD library
- Written in C++
- Imports a general strategy as **CuDD** BDD (written to a file)
- Postprocessing/verification: ABC and NuSMV (BMC)
Modes of operation

Any combination of ...
- DNF/CDNF leaning or their dual case, CNF/DCNF learning
- Smart variable order selection on/off

Best modes found
- CDNF learning + smart variable order selection on
- DNF learning + smart variable order selection on
Circuit sizes (CDNF)

![Graph showing circuit sizes comparison between Unbeast and Ratsy]

- **Circuit Size [#standard cells]**
- **Unbeast**
- **Ratsy**

**Legend:**
- Unbeast
- Ratsy

**Axes:**
- Bshouty (Y-axis)
- Ratsy/Unbeast (X-axis)

**Scale:**
- 10 times smaller
- 100 times smaller

**Data Points:**
- Scattered points representing circuit sizes for different complexity levels.
Circuit depths (CDNF)

Learning

Old/Native

x = RATSY  o = UNBEAST
Computation times

![Graph showing computation times for Old/Native and Learning times, with symbols x = RATSY and o = UNBEAST]

R. Ehlers, R. Könighofer, G. Hofferek
Symbolically Synthesizing Small Circuits
FMCAD 2012
Learning-based implementation extraction:

- an efficient and effective way to compute circuits in reactive synthesis
- not restricted to BDDs, but can be used with any symbolic data structure
- better suited for the task than all other techniques that we tried

Implementation available (for CuDD)

The URL is in the paper!


